International Journal of Research in Advanced Engineering and Technology
Online ISSN: 2455-0876
www.engineeringresearchjournal.com
Volume 1; Issue 3; December 2015; Page No. 67-69

Digital signal processing and its implementation

Shalender Kumar

MCA, Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India

Abstract

High Performance Computing is not the exclusive domain of computational science. Instead, high computational power is required in many devices, which are not built with the primary goal of providing their users with a computer of any kind, but to offer a service in which a powerful computer plays a central role. Medical imaging is an example of the application of such a High Performance Embedded System.

As signals from an X-ray or magneto-resonance device come in at a very high rate, they are processed by a computer to provide the radiologist with a visualization suitable for further diagnosis. Other examples include radar and sonar processing, speech synthesis and recognition, and a broad range of applications in the fields of multimedia and telecommunications.

Keywords: digital signal, visualization, parallelism

Introduction

From the constraints set by the DSP application domain arise some (partially mutually exclusive) requirements for signal processors distinct to those of general purpose processors. DSPs have to be able to deliver enough computational power to cope with demanding applications like image and video processing whilst meeting further constraints such as low cost and low power.

As a result, DSPs are usually highly specialized and adapted to their specific application domain, but notoriously difficult to program. DSPs find application in a broad range of different signal processing environments, which are characterized by their algorithm complexity and predominant sampling rates.

DSP applications have sampling rates that vary by more than twelve orders of magnitude (Glossner *et al*., 2000) [6]. Weather forecasting on the lower end of the frequency scale has sampling rates of about 1/1000Hz, but utilizes highly complex algorithms, while demanding radar applications require sampling rates over a gigahertz, but apply relatively simple algorithms.

Both extremes have in common that they rely on high-performance computing systems, possibly based on DSPs, to meet the timing constraints imposed on them. With the current state of processor technology, it is still not possible to deliver the required compute power for some applications with just a single DSP, but the combined power of several DSPs is needed. Unfortunately, such multi-DSP systems are even more difficult to program than a single DSP.

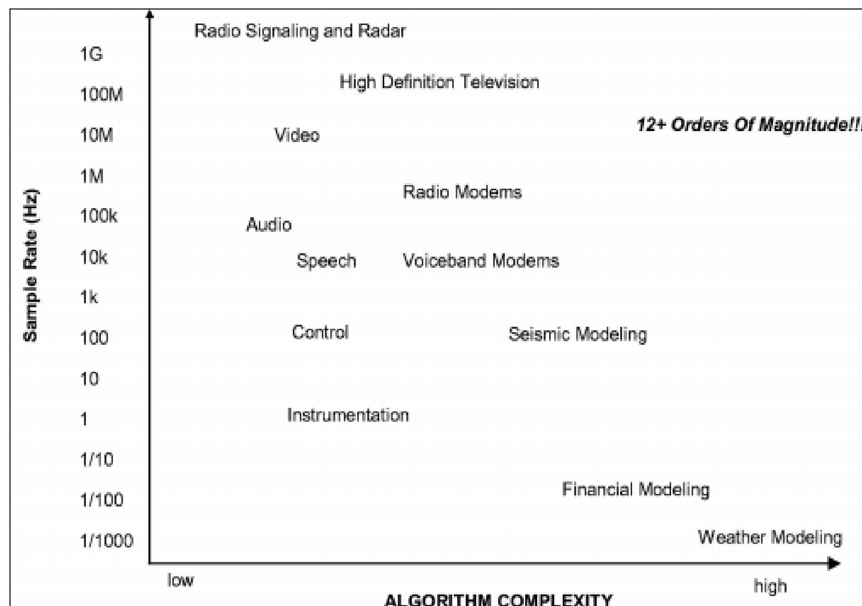

Fig 1: DSP application complexity and sampling rates

DSP and multimedia algorithms are often highly repetitive as incoming data streams are uniformly processed. This regularity suggests that DSP and multimedia applications contain higher levels of parallelism than general purpose

applications, possibly at different granularities.
Figure 1 shows the inherent parallelism of three classes of workloads (general purpose, DSP, video). DSP and video codes contain larger amounts of exploitable parallelism than general purpose codes, with video codes containing the most parallelism. This fact not only simplifies the work of automatically parallelizing compilers, but more importantly it provides the basis for larger performance benefits according to Amdahl's Law.
While general purpose codes can only experience theoretical speedups of up to 10 due to parallel execution, DSP and multimedia codes are subject to more than an order of magnitude higher performance improvements. In the past, DSP software was mainly composed of small kernels and software development in assembly language was acceptable. Similar to other fields of computing, code complexity in the DSP area began to increase and application development using high-level languages such as C became the norm. (Abdelrahman *et al*., 2014) [1]
Recent DSP applications require ten thousand or more lines of C code. Problems exploiting the parallelism in DSP codes arise from this use of C as the dominating high-level language for DSP programming. C is particularly difficult to analyze due to the large degrees of freedom given to the programmer.
Even worse, C permits a low-level, hardware-oriented programming style that is frequently used by embedded systems programmers to manually tune their codes for better performance. Without accurate analyses, however, success in detection and exploitation of program parallelism is very limited. Against this background, optimizing as well as parallelizing compilers must find a way to cope with idiosyncracies of the C programming language and the predominant programming style in order to be successful.

Digital signal processing
Limitations of analogue signal processing operations and the rapid progress made in the field of Very Large Scale Integration (VLSI) led to the development of techniques for Digital Signal Processing (DSP). To enable DSP, an analogue signal is sampled at regular intervals and each of the sample values is represented as a binary number, which is then further processed by a digital computer (often in the form of a specialised Digital Signal Processor (DSP). (Anderson *et al*. 2013) [2]
In general, the following sequence of operations is commonly found in DSP systems (Mulgrew *et al*., 1999) [8]:

- Sampling and Analogue-to-Digital (A/D) conversion.
- Mathematical processing of the digital information data stream.
- Digital-to-Analogue (D/A) conversion and filtering.

From a compiler writer's point of view, it is not necessary to understand how these algorithms work. However, it is important to know and to understand the characteristics of the algorithms and their concrete implementations. These are the inputs supplied to a compiler, and affect the ability of the compiler to generate efficient code.
The high frequency at which multiply-accumulate operations are found in many DSP algorithms has led to the integration of highly efficient Multiply-Accumulate (MAC) instructions in the instruction set of almost all DSPs. (Andreyev *et al*. 2012) [3]
MAC operations typically take two operands and accumulate the result of their multiplication in a dedicated processor register. Thereby, sums of products can be implemented using very few, fast instructions. Further improvements come from Zero-overhead loops (ZOLs). A loop counter can be initialized to a constant value which then determines how often the following loop body is executed.
This eliminates the need for potentially stalling conditional branches in the implementation of loops with fixed iteration counts. Streaming data as the main domain of DSP shows very little temporal locality. This and the real-time guarantees required from many DSP systems make data caches unfavorable.
Instead, fast and deterministic on-chip memories are the preferred design option. Memory in DSPs is usually banked. Two independent memory banks and internal buses allow for the simultaneous fetch of two operands as required by many arithmetic operations, e.g. MAC.
Address computation is supported by Address Generation Units (AGUs), which operate in parallel to the main data path. Thus, the data path is fully available for user calculations and does not need to perform auxiliary computations. (Balasa *et al*. 2014) [4]
In embedded systems processors usually work under tighter constraints than in a desktop environment. This is particularly true for DSPs, which are often faced with real time performance requirements on top of other system constraints. DSP system engineering is not the issue of this thesis. However, it is important to understand the main system requirements to avoid solutions that are feasible on their own, but do not fit into the overall system design.
For example, compiler transformations that blow up code size to such an extent that it does not fit into the restricted on-chip memories differentiate embedded compiler construction from general-purpose compilers where code size is less critical.

Software design and implementation
The DSP software design process has the peculiar property of being split into two separate high-level and low-level stages. On the high level, simple algorithms are formulated by means of equations which form basic blocks for the construction of more complex algorithms.
These high-level formulations are translated into Synchronous Data Flow (SDF) Graphs and implemented in high-level languages like Matlab. Due to performance reasons, proven high-level implementations are re-implemented on a lower level using programming languages like C or C++ enhanced with system specific and non-standard features. (Duesterwald *et al*. 2013) [5]
Where performance is still not sufficient, assembly is used to optimize performance bottlenecks. In this work, the lower level of abstraction is considered. Programmers are provided with an optimizing and parallelizing C compiler, which saves him from manually tuning and parallelizing code for specific target architecture.

DSPS and multimedia processors
Leupers (2010) [7] indentifies five classes of embedded processors: Microcontrollers, RISC processors, Digital Signal Processors (DSPs), Multimedia processors and Application Specific Instruction Set Processors (ASIPs). Of these five classes, only DSPs and multimedia processors are of interest as the targeted application domain covers DSP and multimedia

workloads.
According to him, DSPs are characterized by special hardware to support digital filter and Fast Fourier Transform (FFT) implementation, a certain degree of instruction-level parallelism, special-purpose registers and special arithmetic modes. Multimedia processors, on the other hand, are specially adapted to the higher demands of video and audio processing in that they follow the VLIW paradigm for statically scheduling parallel operations.
To achieve a higher resource utilisation multimedia processors often offer SIMD instructions and conditional instructions for the fast execution of if-then-else statements. However, manufacturers have not generally adopted this classification and tend to classify and name their products by the type of applications found in the market they are aiming at.
In particular, manufacturers refer to their processors aiming at multimedia processing as DSPs, too. We adhere to the manufacturers' classification (DSP/multimedia processor) of their processors. Digital signal processors as specialized processor architectures have a memory system which significantly differs from those found in general-purpose processors and computing systems

Significance of the study
Most embedded DSPs comprise of several kilobytes of fast on-chip SRAM. This is due to the fact that SRAM integrated on the same chip as the core processor allows for fast access without wait states. Thus, processor performance is not impeded by the memory system. Furthermore, on-chip SRAM allows for the construction of inexpensive and compact DSP systems with a minimal number of external components.
Usually, a DSP's internal memory is banked, i.e. distributed over several memory banks, thereby allowing for parallel accesses. The reason for this physical memory organization comes from the fact that many operations in DSP applications require two or sometimes three operands to compute a single result.
Fetching these operands sequentially leads to poor resource utilization as the processor might have to wait until all operands become available before it can resume computation. Parallel accesses to operands are a way of matching processor and memory speed by providing higher memory bandwidth. Hence, appropriate assignment of program variables to memory banks is crucial to achieve good performance.
The on-chip storage capacity is not always sufficient to hold a program's code and data. In such a case, external memory can be connected to a DSP through an external memory interface. Often the latency of external memory is higher than that of the on-chip SRAM as a cheaper, but slower memory technology might be used (lower cost and improved memory density).
Additionally, bandwidth to external memory is usually smaller as parallel internal buses are multiplexed onto a single external bus (smaller pin count) operating at a slower clock rate (simpler board design, cheaper external components). Avoiding excessive numbers of external memory accesses by appropriate program/data allocation and utilization of on-chip memory together with the exploitation of efficient data transfer modes (e.g. Direct Memory Access (DMA)) are necessary to save program performance from severe degradation.

References

1. Abdelrahman Exploiting task-level parallelism using pTask. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '14), Sunnyvale, CA, USA, 2014, 252-263.
2. Anderson. Data and computation transformations for multiprocessors. In Proceedings of 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '13), Santa Barbara, CA, USA, 2013, 166-178.
3. Andreyev The technique of high-level optimization of DSP algorithms implementation, 2012.
4. Balasa. Transformation of nested loops with modulo indexing to affine recurrences. Parallel Processing Letters 2014; 4(3):271–280.
5. Duesterwald. A practical data flow framework for array reference analysis and its use in optimizations. In Proceedings of the SIGPLAN Conference on Programming Languages Design and Implementation (PLDI '13), Albuquerque, NM, USA, 2013, 68-77.
6. Glossner. Analysis of high-level address code transformations for programmable processors. In Proceedings of Design and Test in Europe Conference (DATE '10), Paris, France, 2000, 9-13.
7. Leupers. Detecting coarse-grain parallelism using an interprocedural parallelizing compiler. In Proceedings of the 2014 ACM/IEEE International Conference on Supercomputing (ISC '95), San Diego, CA, USA, 2010.
8. Mulgrew. An approach for automated application of platform-dependent source code transformations, 1999.